

# NWEN 243 Networked Applications

## Lab 6: Building a TCP Server

### Objectives

- Experience using TCP
- Achieve the completion of a TCP server

### Requirements

- This is an individual lab written in C.
- We will be writing programs that you execute from the shell command line.
- Your server will need to handle multiple simultaneous connections. In C, we will use `fork()` to create a new process to service each connection. You will be supplied the code for this in the skeletons.
- You are recommended (but not compulsory) to demonstrate your work to your lab tutor AND you must submit your program through the online submission system.

### Preliminaries

- TCP – the transmission control protocol is a reliable byte ordered transport layer service. Delivery and order are guaranteed.
- To use TCP you will need to use the Socket API. The socket API is modeled on the file system, so uses *read/recv* and *write/send* to access the network.
- You should use your TCP client developed in Lab 5 to connect to and to test your server. Also, you should MODIFY your TCP client to include loops so that a series of 100 requests can be sent in sequence to the TCP server. Please make sure to introduce at least 3 seconds of delay before next iteration of the loop (i.e. sending next request to the TCP server).
- You should run 5 TCP clients (with loops) concurrently to ensure your TCP sever works properly (can handle all requests from all TCP clients correctly).

### The Exercise

Your task is to complete a TCP server program that will

1. Wait for a client to connect.
  - a. Fork a worker process
    - i. Service the client using the connection socket.
    - ii. Close the connection socket and exit process
  - b. Loop back around for the next connection.

What will your server do? You can duplicate the SHOUTING server if you wish, but you may also implement some other service if you wish – such as serving a file (a little dangerous) or reversing the client's string, or anything else simple – this is not the focus of the exercise.

## Resources

- There is a C skeleton for TCP server provided for this exercise.
- Look at our tutorial notes on socket programming. You can find in the notes a diagram detailing the steps that a TCP server takes as well as sample code for building concurrent servers. Use this information to guide your program.

## Run your Server

Use:

```
%> ./server portnum
```

Note that we take the port number as a command line argument, this is because you need to specify it for the client – you could use a random one, but it is easier when testing with your client to use a fixed port number. Make sure you check the port is not already used, and kill old servers that you have left sitting around.

## Topic questions (1/2 mark each)

Please include your answers to these questions in a separate PDF document, and ensure you submit it at the same time as your project.

**Q1.** Explain the concept of *out-of-band data* in socket communication. Will out-of-band data always be delivered reliably?

**Q2.** Is it possible to force a socket to empty its data in the buffer? If so, how to do that?

**Q3.** Explain when SYN segment is used in TCP communication. Can any SYN segment carry data payload and why?

**Q4.** Consider hosts A and B communicating over a TCP connection. Assume unrealistically that the initial sequence number for each of A and B is 0 (after the handover process). Assume that all segments sent between A and B have 20 byte headers. A sends B a segment with a 100 byte payload, B responds with a segment with a 100 byte payload and then another segment with a 200 byte payload, and finally A responds with a segment with a 50 byte payload. Give the value of the sequence number field and acknowledgement number field for each segment.

**Q5.** Suppose that a socket client running on an ARM processor is receiving an integer from a socket server running on an INTEL processor. Suppose also that the ARM processor stores an integer in 32 bits and the INTEL processor stores an integer in 64 bits. Can the socket client receive the correct integer from the socket server? Make sure that you explain your answers.

## Hand in

1. You are recommended to demonstrate your working program to your lab tutor.
2. You should also submit your FULLY COMMENTED server code. Comments

- must be added to make it clear that we understand what is going on.
3. No lab report is needed.
  4. Include in your submission a PDF document with your answers to the topic questions.
  5. Online submission is due on **Sunday 1 October 2017 at 23:59**.
  6. Marking (total of 10%)
    - a. The questions are worth **2.5 marks**.
    - b. The working code is worth **7.5 marks** (see below for more details on grading for the code)

### Grading for the code

- C grade: the TCP server program can successfully support a single TCP client.
- B grade: the TCP server program can successfully support 5 concurrent TCP clients.
- A grade: fully commented submitted code shows your good understanding of TCP server code. Also the TCP server code is implemented to a high standard (well formatted/structured, careful error handling and reporting, well commented).

### Useful stuff for C

*int socket(int domain, int type, int protocol);*

socket() creates an endpoint for communication and returns a socket descriptor. The domain parameter specifies a communication domain; this selects the protocol family which will be used for communication. You should use AF\_INET. This is the same for a server and a client.

*int bind(int sockfd, const struct sockaddr \*my\_addr, socklen\_t addrlen);*

bind() gives the socket sockfd the local address my\_addr. my\_addr is addrlen bytes long. Traditionally, this is called "assigning a name to a socket." When a socket is created with socket, it exists in a name space (address family) but has no name assigned. It is necessary to assign a local address using bind() before a SOCK\_STREAM socket may receive connections

*int listen(int sockfd, int backlog);*

To accept connections you first need to tell the socket to 'listen'. The backlog parameter defines the maximum length the queue of pending connections may grow to – set this to 5.

*int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);*

The accept() system call waits until there is a connection request, and then extracts the first connection request on the queue of pending connections, creates a new connected socket, and returns a new file descriptor referring to that socket. The original socket sockfd is unaffected by this call.

*int read(int sockfd, void \*buf, int count);*

read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

*int write(int sockfd, const void \*buf, int count);*

write() writes up to count bytes to the file referenced by the file descriptor fd from the buffer starting at buf.

*int close(int fd);*

close() closes a file descriptor, so that it no longer refers to any file and may be reused.

Notes:

1. Many functions return a value  $< 0$  to indicate an error. You should check this.
2. Make sure you don't overflow any buffers, and limit the sizes correctly.